BG/P Software Overview

Brian Smith (Rochester, MN)
smithbr@us.ibm.com

Rajiv Bendale (bendale@us.ibm.com)
Kirk Jordan (kjordan@us.ibm.com)
Jerrold Heyman (jheyman@us.ibm.com)
Bob Walkup (walkup@us.ibm.com)

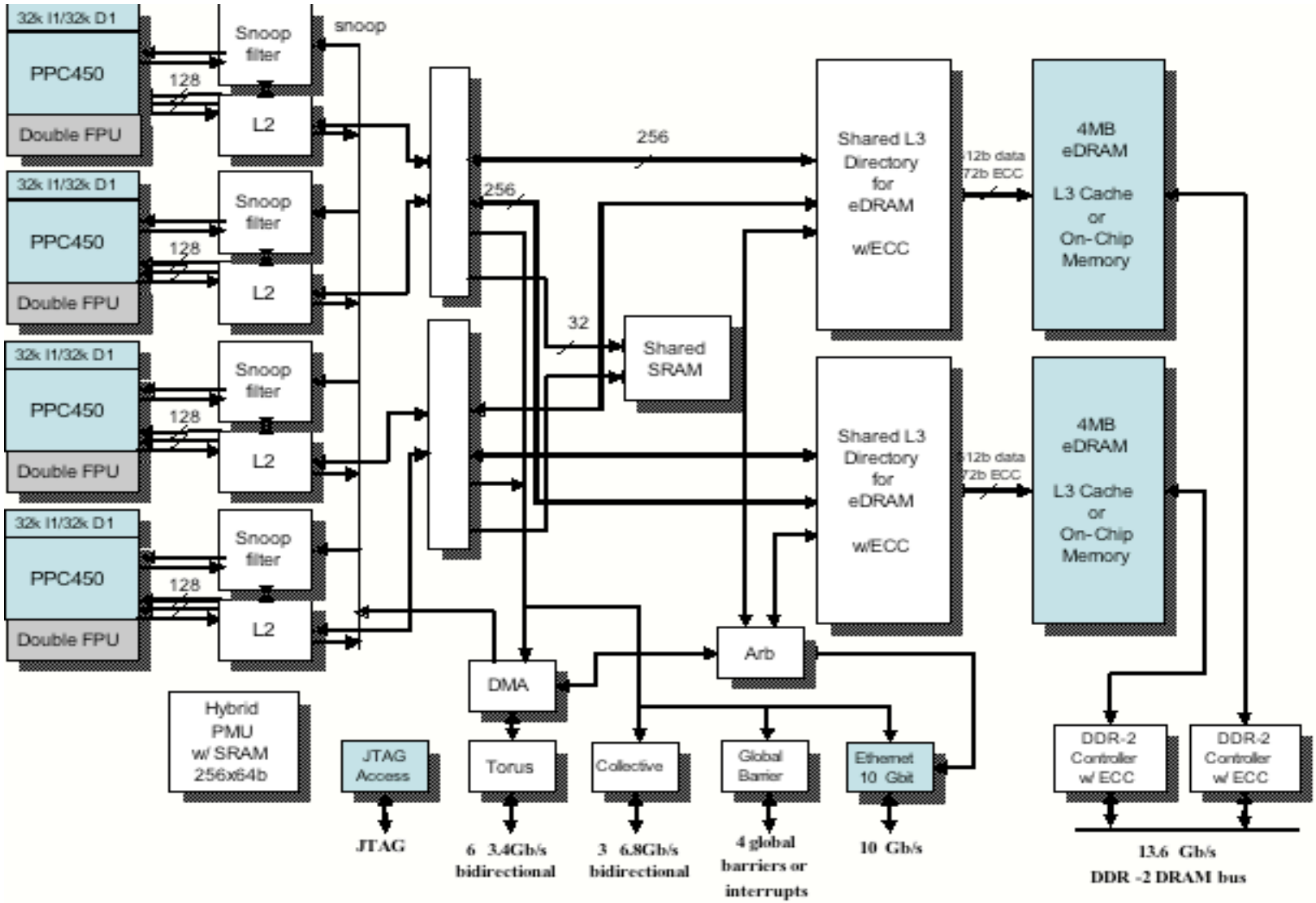
Overview

- **BlueGene/P Quick Introduction**
  - BG/P Hardware
  - Software
- **MPI Implementation on BlueGene/P**
  - General Comments and Optimization Suggestions
- **I/O On BGP**

BG/P Hardware Introduction

- **Up to 256^3 compute processors**
  - Largest machine: 40960 nodes at ANL
  - Relatively slow processors (850 MHz)
  - But -- low power, low cooling, very high density
- **System-on-a-chip technology (4 cores, 8 FPUs, memory controllers, networks, etc on single ASIC)**
- **3 very high-speed application networks**
  - Torus network has a DMA engine

BG/P ASIC



PowerPC 450 Processor

- Offshoot of PPC440 Processor
- 32-bit architecture at 850 MHz
- Single integer unit (fxu)
- Single load/store unit
- Special double floating-point unit (dfpu)
- L1 Data cache : 32 KB total size, 32-Byte line size,
  - 64-way associative, round-robin replacement
  - 4 cores on BG/P are L1 cache coherent
- L2 Data cache : prefetch buffer, holds 16 128-byte lines
- L3 Data cache : 8 MB
- Memory : 2 GB DDR, ~13.6GB/s bandwidth
- Double FPU has 32 primary floating-point registers, 32 secondary floating-point registers, and supports :
  - standard powerpc instructions, which execute on fpu0 (fadd, fmadd, fadds, fdiv, …), and
  - SIMD instructions for 64-bit floating-point numbers (fpadd, fpmadd, fpre, …)
- Floating-point pipeline : 5 cycles

BG/P Hardware

- **Double Hummer FPUs**
  - 2 64bit FPUs
  - Not independent though
  - Requires careful alignment considerations
  - Compilers are good now, but hand-tuning critical sections might be necessary/valuable

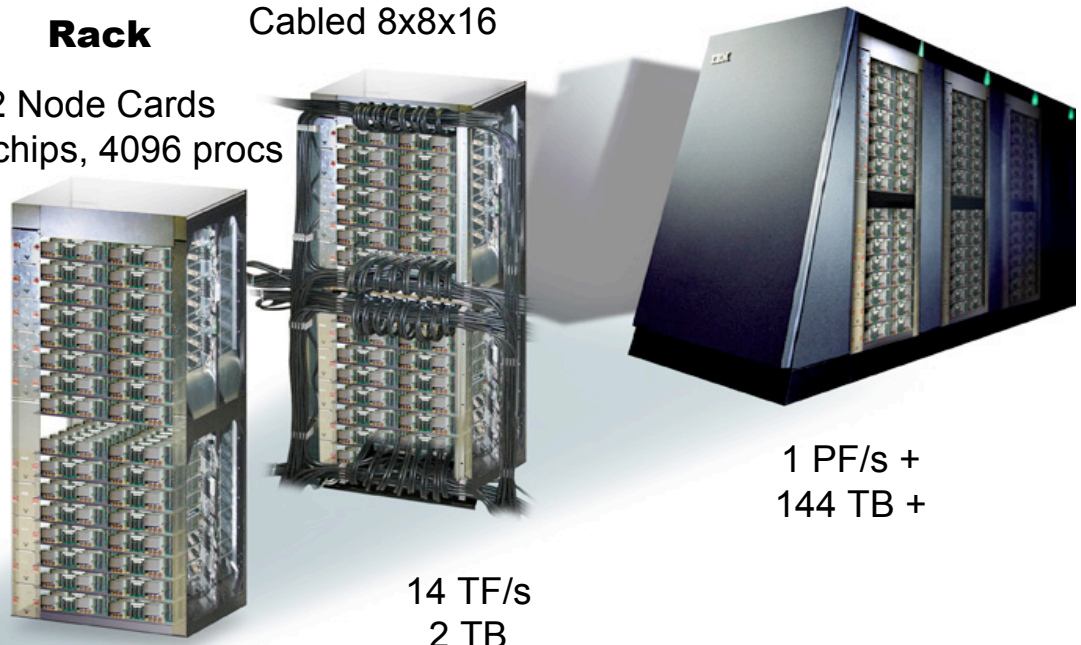
Mandatory Scaling Slide

Blue Gene/P

System
1 to 72 or more Racks

Cabled 8x8x16

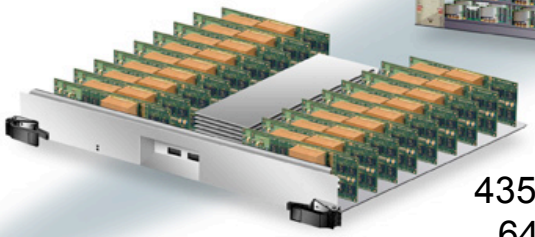
Rack
32 Node Cards
1024 chips, 4096 procs



1 PF/s +
144 TB +

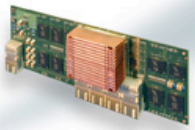
14 TF/s
2 TB

Node Card
(32 chips 4x4x2)
32 compute, 0-2 IO cards



435 GF/s
64 GB

Compute Card
1 chip, 20 DRAMs



13.6 GF/s
2.0 GB DDR
Supports 4-way SMP

Chip
4 processors

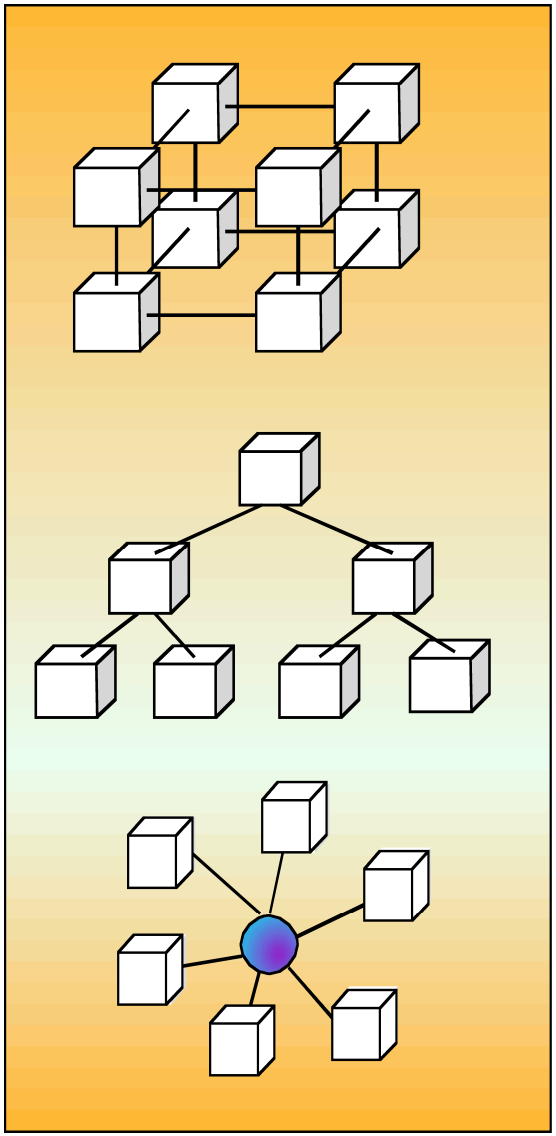
13.6 GF/s
8 MB EDRAM

Performance (June 2008 Top 500)

- **4 of the top 20 machines are BlueGene/P**

Racks	Installs	Ranking
40	1	3
16	1	6
10	1	8
8	1	13
4	1	37
3	2	51
2	2	74

Blue Gene/P Interconnection Networks



3 Dimensional Torus

- Interconnects all compute nodes
  - Communications backbone for computations
- Adaptive cut-through hardware routing
- 3.4 Gb/s on all 12 node links (5.1 GB/s per node)
- 0.5 µs latency between nearest neighbors, 5 µs to the farthest
  - MPI: 3 µs latency for one hop, 10 µs to the farthest
- 1.7/2.6 TB/s bisection bandwidth

Collective Network

- Interconnects all compute and I/O nodes
- One-to-all broadcast functionality
- Reduction operations functionality
- 6.8 Gb/s of bandwidth per link
- Latency of one way tree traversal 2 µs, MPI 5 µs

Low Latency Global Barrier and Interrupt

- Latency of one way to reach all 72K nodes 0.65 µs, MPI 1.6 µs

Other networks

- 10Gb Functional Ethernet
  - I/O nodes only
- 1Gb Private Control Ethernet
  - Provides JTAG access to hardware. Accessible only from Service Node system

BG/P Software

- **Compute Node Kernel (CNK)**
  - Minimal kernel – handles signals, function shipping syscalls to I/O nodes, starting/stopping jobs, threads
  - Not much else
  - Very “linux-like”, uses glibc
    - Missing some system calls (fork() mostly)
      - Limited support for mmap(), execve()
    - But, most apps that run on Linux work out-of-the-box on BG/P

MPI on BG/P – Run modes

Virtual Node Mode (VNM)

- **Each core gets its own MPI rank, kernel image**
- **4x the computing power**
- **Not necessarily 4x the performance**
  - Each core gets ¼ memory
  - Network resources split in fourths
  - Cache split in half (L3) and 2 cores share each half
  - Memory bandwidth split in half
  - CPU does compute and communication, though DMA helps
  - Global communication can be expensive
  - No threads

SMP

- **Full memory available**
- **All resources dedicated to single kernel image**
- **Can start up to 4 pthreads/OpenMP threads**
  - OMP_NUMTHREADS=x

Dual

- **Hybrid of the two modes**
- **2x MPI ranks of SMP, each rank can start 1 additional thread**
- **½ memory of SMP per rank**

BG/P Software

- **Compilers**
  - IBM XL compilers (f77, f90, C, C++)
    - Latest version 11.1 for fortran and 9.0 for C/C++
  - GNU (gcc, g++, gfortran) also available
  - IBM ESSL libraries optimized for BG/P
    - Good community success with gotoBLAS, ATLAS
  - MASS(V) libraries
  - Applications are built on the front-end nodes via cross compiling

BG/P Software

- **Control System**
  - Runs on service node
  - Compute nodes are stateless
    - State information is stored in db2 databases
  - Database also monitors performance, environmentals, etc
  - Boots blocks, monitors jobs, etc
  - Interaction via Navigator, LoadLeveler, etc.

BG/P I/O Nodes

- **Scalable Configurations: Compute / IO Node ratio**
  - 16, 32, 64, 128 to 1.
- **IO node specs**
  - Max bandwidth per IO Node = 1250 MB/s (10 Gb/s Ethernet)
- **Streaming IO (Sockets) performance**
  - We've seen 500+ MB/s but we don't have a good test environment in Rochester
  - IO can scale linearly due to parallel IO
- **CIOD environment variables to fine tune file system performance**
  - CIOD_RDWR_BUFFER_SIZE
    - Should be set to the GPFS block size (typically 2MB or 4MB)
  - Really only set-able by sysadmins. Need to set up the IO node ramdisk image to have the env var

BG/P Software

- **I/O Node Kernel**
  - Linux (MCP)
  - Very minimal distribution (almost everything on the I/O node is in busybox)
  - Only connection from compute nodes to outside world
  - Handles syscalls (ie fopen()) and I/O requests

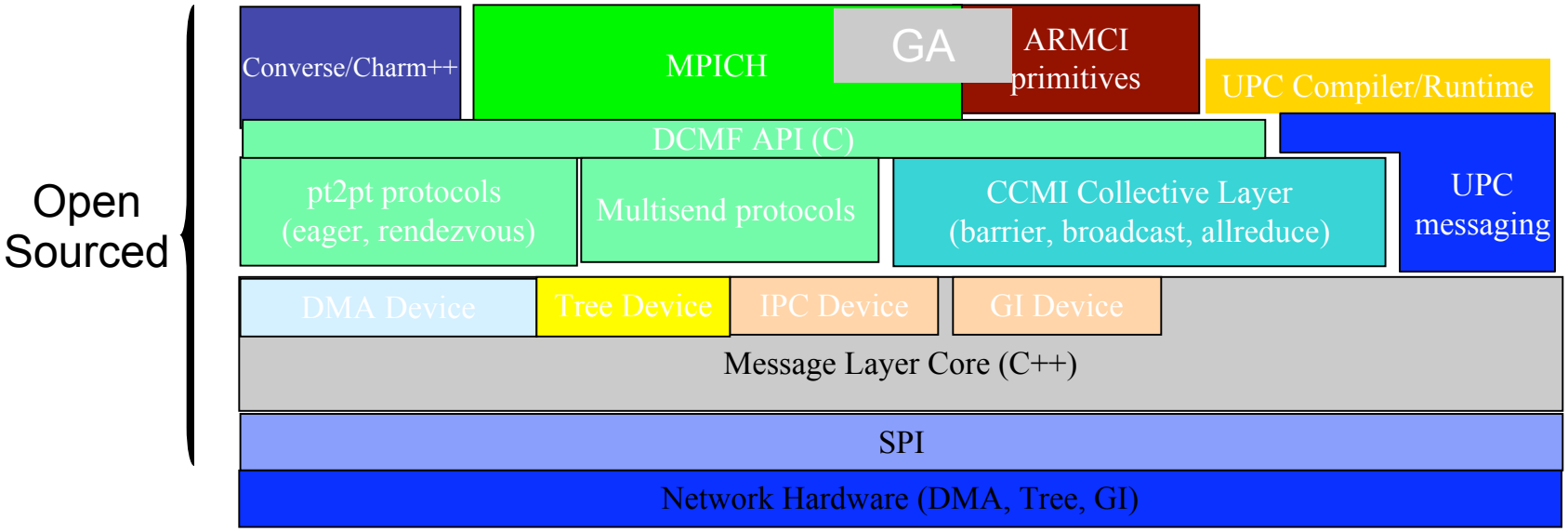
BGP Communications Overview

- **The stack:**
  - SPI
  - DCMF/CCMI
  - MPI, GA/ARMCI
- **Optimizations**

BG/P Communications Software Stack Design

- **Support many programming paradigms**
- **Non blocking communication**
  - Support for asynchronous communication where possible
- **Open source messaging runtime**
  - Extendible
  - Component oriented design
  - http://dcmf.anl-external.org/wiki
  - Mailing list
- **General Availability will have product version of software**
  - Extensions provided through contribs

BGP Messaging Stack



- **Multiple programming paradigms supported**
  - MPI and ARMCI, Charm++ and UPC (as research initiatives)
- **SPI : Low level systems programming interface**
- **DCMF : Portable active-message API**

MPI on BG/P – SPI

- System programming interface
- Very low level, basically right on top of the hardware
- Use is complicated, but can provide the best possible performance
- Very stable interface.
- Doxygen comments, plus look at higher levels for examples
- Used by DCMF, some QCD codes

MPI on BG/P - DCMF

- **Non blocking runtime**
- **Multiple Protocol Registration**
- **Active Messaging API**
- **Essentially just point-to-point interfaces:**
  - DCMF_Send, DCMF_Put, DCMF_Get
  - DCMF_Multisend
- **DCMF_Messager_advance()**
  - Call handlers
  - Call callbacks when the counters have hit zero
- **Heavily doxygenated, lots of usage examples**
- **Good performance/complexity tradeoff for applications**
- **Portable**
  - Sockets interface available soon, currently running on a number of platforms
- **Fairly stable interface (one or two minor changes on the horizon for BGP V1R3)**
- **Proposed BoF for SC08**
- **Paper at ICS08**

MPI on BG/P - CCMI

- **Component Collective Message Interface**
  - Portable layer for collectives
    - Basically requires a multi-send protocol to implement our optimized collectives
    - We have an MPI multi-send implementation
  - Paper being presented at Euro PVM/MPI
  - Sits on top/to the side of DCMF

MPI on BG/P - MPI

- **The primary function of BG/P: running your MPI-based applications**
- **Our MPI looks suspiciously like MPICH2 1.0.x**
  - "MPI standard 2.0-"
  - No process management (MPI_Spawn(), MPI_Connect(), etc)
  - Based on MPICH2 1.0.7 base code
  - We are working closely with ANL to re-integrate all BGP changes in their main tree

MPI on BG/P – GA/ARMCI

- **ARMCI – Aggregate remote memory copy interface**
- **From PNNL**
- **Sits on top of DCMF and MPI**
- **Very good performance, close to straight DCMF code**
- **Paper being presented at ICPP in Seattle**

MPI on BG/P – GA/ARMCI

- **Global Arrays from PNNL**
- **Used by major applications (NWChem, GAMESS-UK, ScalaBLAST, gp-shmem, GAMESS-US, etc)**
- **Sits on top of ARMCI and MPI**
- **Support for large distributed arrays**
- **Considered "Technology Preview"**
  - We are working closely with PNNL to improve support of GA/ARMCI on BGP (performance)

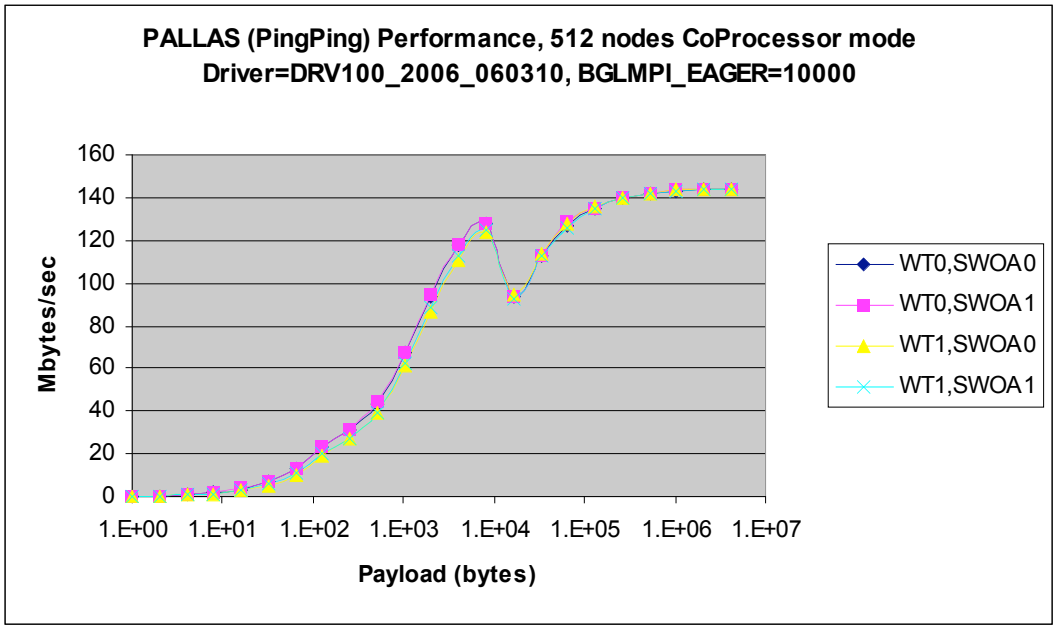
Communications Optimizations

MPI on BG/P – Optimization Suggestions

- **Point-to-point**
  - DMA Tuning
- **Mapping**
- **Collectives**
  - Our strategies

MPI on BG/P – Point-to-Point

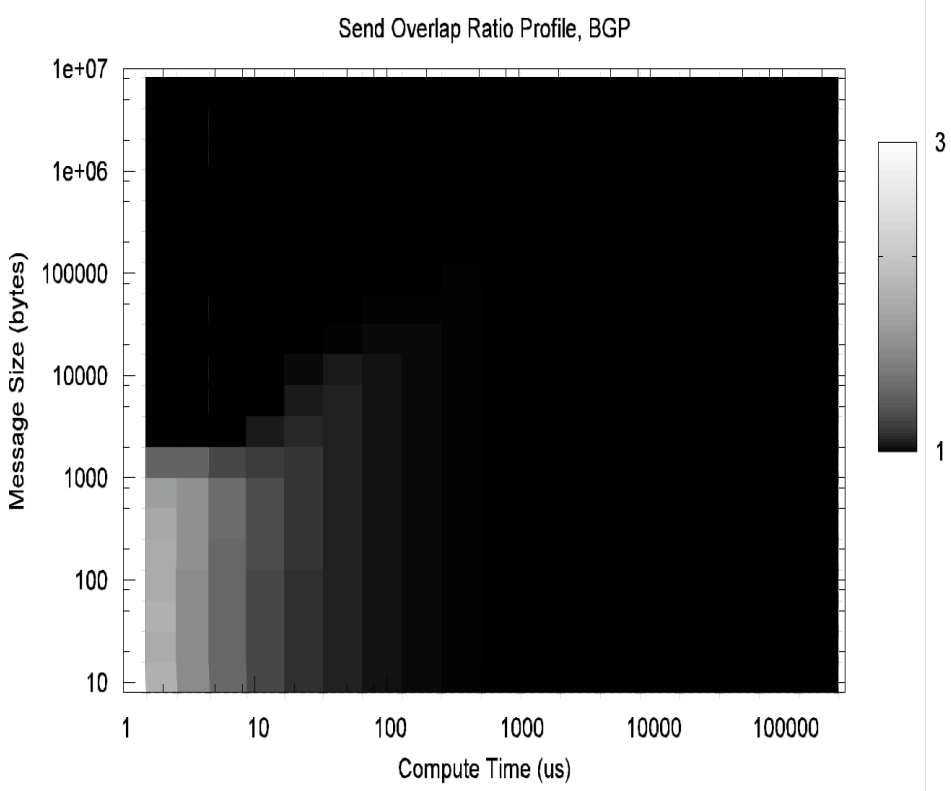
- **Change rendezvous messaging size**
  - Larger partitions should have lower cutoff
  - Increase cutoff if mostly nearest-neighbor communications
  - Environment variable: DCMF_EAGER=xxxxx or DCMF_RVZ=xxxxx or DCMF_RZV=xxxxx
  - Default: 1200 bytes

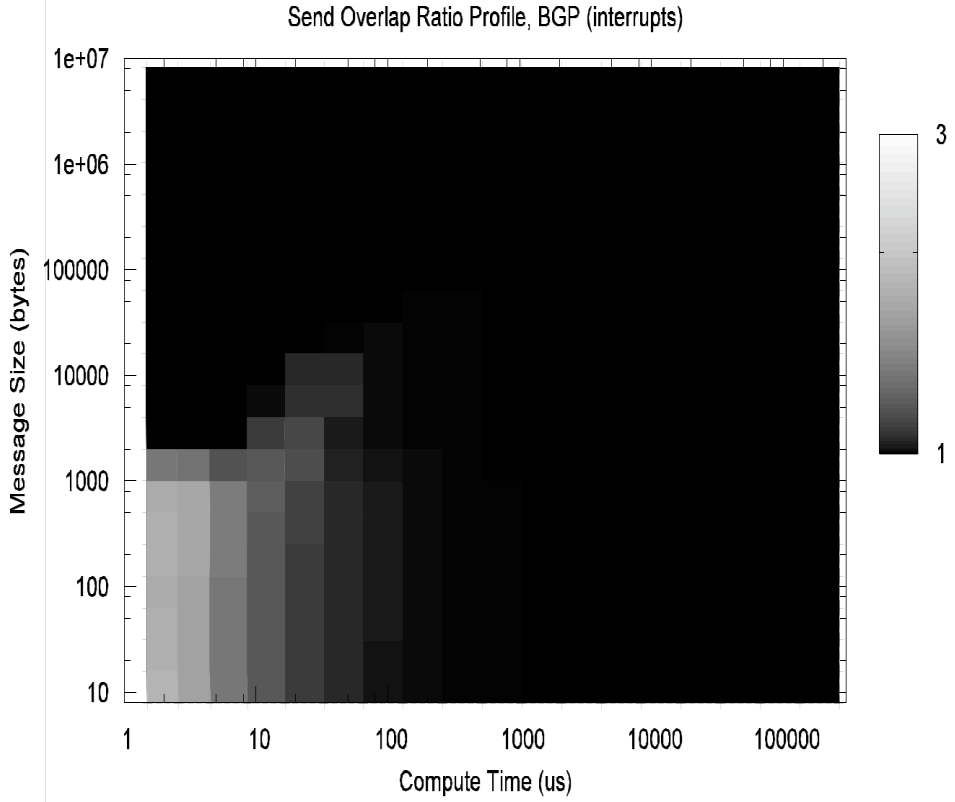


MPI on BG/P – Point-to-Point

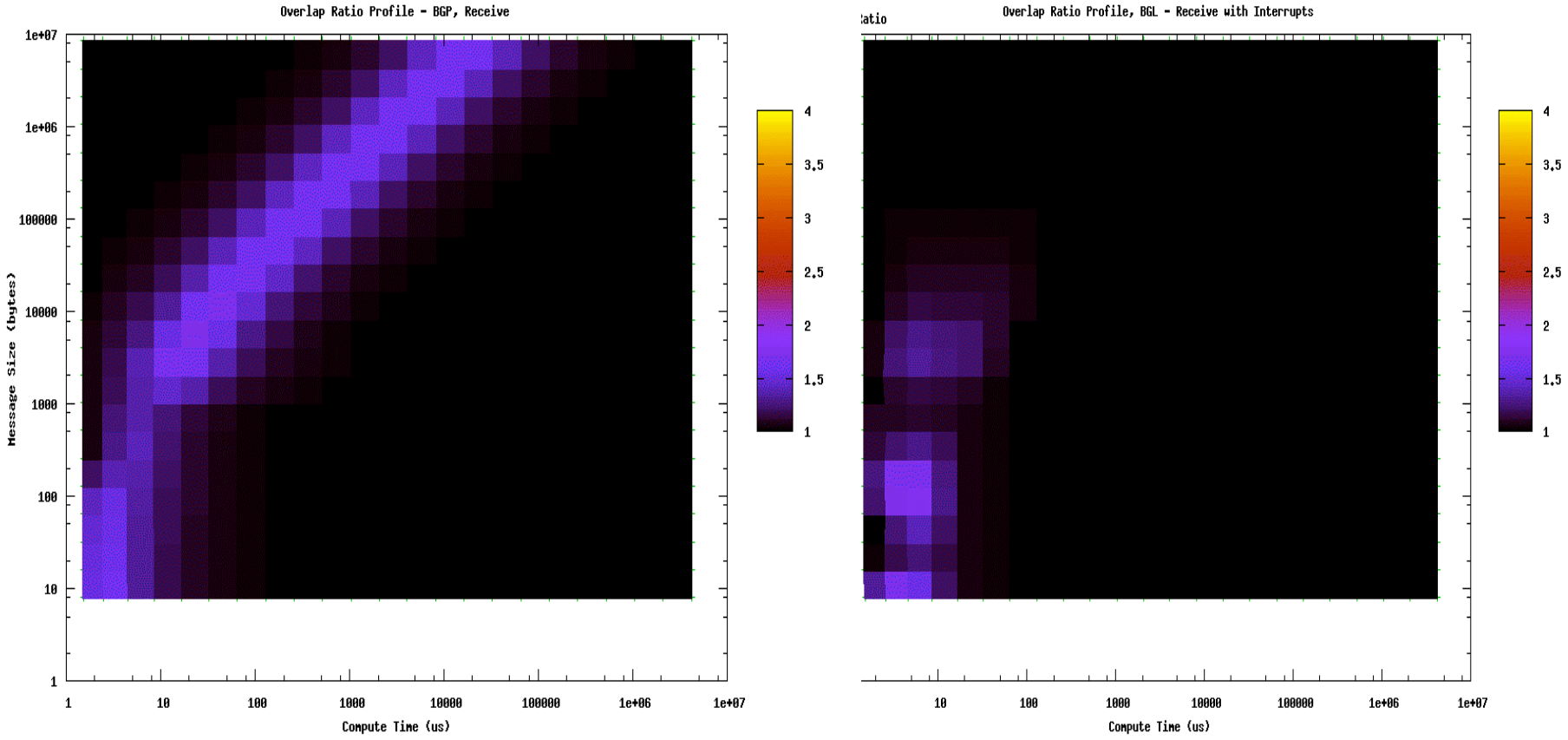
- **Overlapping communication and computation:**
  - Easier on BG/P than BG/L
  - Keep programs in sync as much as you can
    - alternate computation and communication phases
  - Post receives/waits early and often
  - Try interrupts
    - DCMF_INTERRUPTS=1
    - Very narrow region of overlap
- **Processors are slow relative to network speed**

Overlap of Computation and Communication





Overlap with Interrupts



MPI on BG/P – Point-to-Point

- **Avoid load imbalance/“master node”**
  - bad for scaling
- **Shorten Manhattan distance messages have to traverse**
  - send to nearest neighbors!

MPI on BG/P – Point-to-Point

- **Avoid synchronous sends**
  - increases latency
  - Sometimes required to prevent unexpected messages and memory problems
    - Usually best to rethink
- **Avoid buffered sends**
  - memory copies are bad and bsend is pointless on this implementation
- **Avoid vector data, non-contiguous data types**
  - BG/P MPI doesn't have a nice way to deal with them (requires at least one memcopy, usually 2) and no BG/P specific optimizations
- **Post receives in advance/often**
  - unexpected messages hurt performance and take memory
- **Be cache friendly: align to 16 byte (32 byte is even better)**
  - More in compiler talk, but __alignx() and disjoint pragmas.

MPI on BG/P – DMA Tuning Parameters

- **DCMF_RECFIFO=xxxx**
  - Default 8mb
  - Size (in bytes) of each DMA reception FIFO
  - Larger values can reduce torus network congestion, but takes application memory
  - Note: In VNM this is 8mb PER RANK
- **DCMF_INJFIFO=xx**
  - Default is 32k
  - Size (in bytes) of each DMA injection FIFO
  - Larger values can help reduce overhead when there are many outstanding messages
  - DCMF messaging uses up to 25 injection FIFOs.
  - Rounded up to a multiple of 32 (each descriptor is 32 bytes)

MPI on BG/P – DMA Tuning Parameters

- **DCMF_RGETFIFO=xx**
  - Default 32k
  - Size (in bytes) of the remote get FIFOs.
  - Larger values can help reduce overhead when there are many outstanding messages
  - DCMF messaging uses up to 7 remote get FIFOs.
  - Rounded up to the nearest multiple of 32
- **DCMF_POLLLIMIT=x**
  - Default 16
  - This sets the limit on the number of consecutive non-empty polls of the reception FIFO before exiting the advance function
  - A value of 0 means stay in advance until the FIFOs are empty

MPI on BG/P – DMA Tuning Parameters

- **DCMF_INJCOUNTER=x**
  - Default is 8
  - Sets the number of DMA injection counter subgroups that DCMF can use. Maximum is 8.
  - Only useful if something else is using the DMA, ie, an application making SPI calls directly
- **DCMF_RECCOUNTER=x**
  - Same as INJCOUNTER, but for reception counter subgroups.

MPI on BG/P – DMA Tuning Parameters

- **DCMF_FIFOMODE=DEFAULT/RZVANY/ALLTOALL**
  - Determines how many injection FIFOs are used and what they are used for
  - DEFAULT uses 22 injection FIFOs.
  - RZVANY uses 6 more remote get FIFOs than DEFAULT.
  - ALLTOALL uses 16 alltoall FIFOs (instead of 6) that can inject into any of the torus FIFOs.
  - Try RZVANY if your app uses lots of large messages
  - Try ALLTOALL if your app does lots of alltoall communications
  - Note: RZVANY and ALLTOALL consume more memory – 32k per extra FIFO
  - Note: You can't coexist with "native" SPI calls if you aren't in DEFAULT mode

MPI on BG/P - Mapping

- **Mapping can help point-to-point based codes**
- **Stock mappings implemented already – XYZT, XZYT, YXZT, YZXT, ZXYT, ZYXT, TXYZ, TXZY, TYXZ, TYZX, TZXY, TZYX**
- **Default mapping in VNM/Dual is XYZT (sorry about that)**
  - Cores on the same node are not contiguous MPI ranks
  - specifying TXYZ can be **very** helpful
- (almost)**Arbitrary mapping files can be used**
- **mpirun –mapfile XYZT**
- **mpirun –mapfile /path/to/my/mapfile.txt**

MPI on BG/P - Mapping

- **Example:**

  0 0 0 0
  0 0 0 1
  0 0 1 0
  0 0 1 1

- **Line number in file is desired MPI rank. Each node in partition must be listed in file.**
- **Coordinates are X, Y, Z, and core ID (virtual node mode/dual mode)**

MPI on BG/P – Communicator Creation

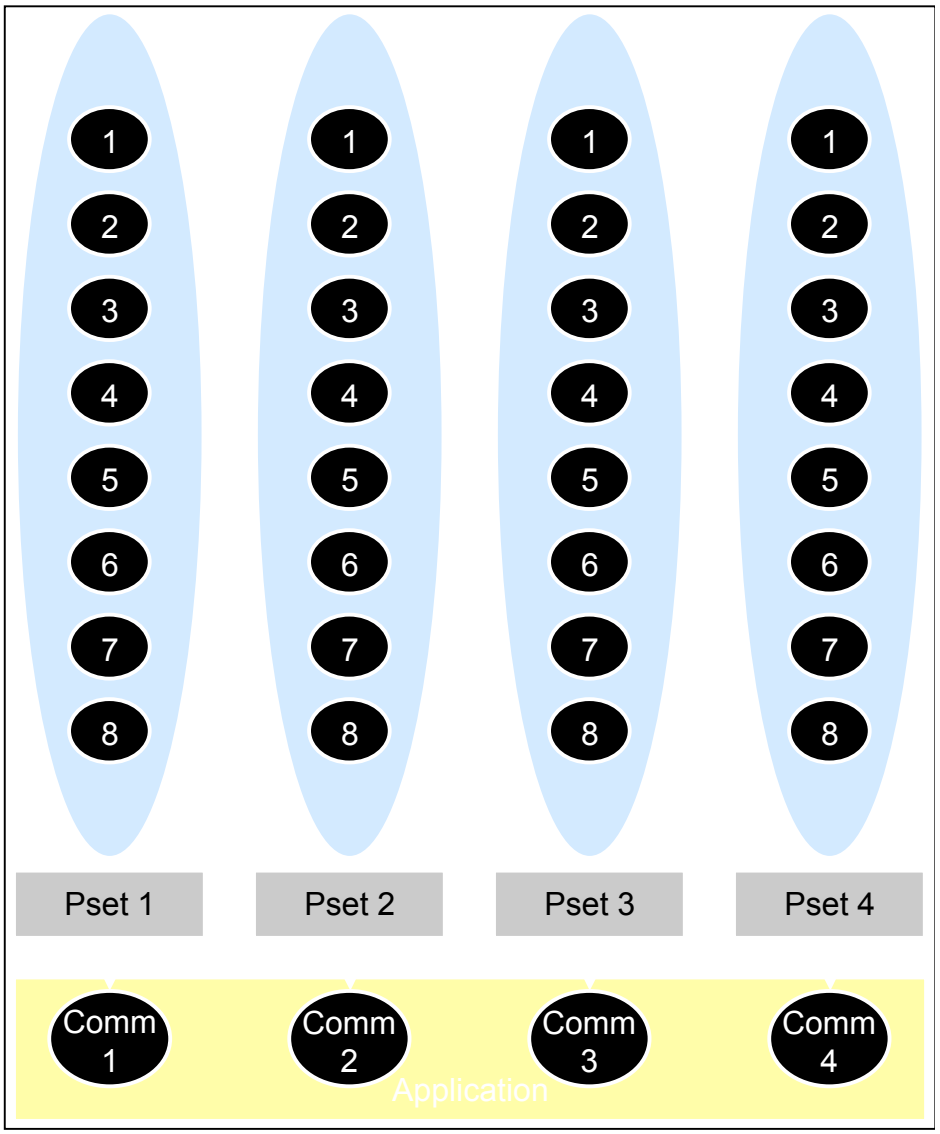
- **APIs to map nodes to specific hardware and/or pset configurations**
  - MPIX_Cart_comm_create()
    - Returns an MPI communicator that is exactly the same as the underlying hardware
    - Eliminates need for complex node-mapping files
  - MPIX_Pset_same_comm_create()
    - Returns a communicator where all nodes belong to the same pset
  - MPIX_Pset_diff_comm_create()
    - Returns a communicator where all nodes have the same pset rank
  - MPI_Cart_create() with reorder true attempts to give communicators that mirror hardware
- **DCMF_TOPOLOGY=0 disables any attempt to give good communicators from MPI_Cart_create() calls**

MPI on BG/P - MPIX_Cart_comm_create()

- **Creates a 4D Cartesian communicator**
  - Mimics the hardware
  - The X, Y & Z dimensions match those of the partition
  - The T dimension will have cardinality 1 in copro, 2 in dual, 4 in VNM
  - The communicator wrap-around links match
  - The coordinates of a node in the communicator match its coordinates in the partition
- **Important**
  - This is a collective operation and must be run on all nodes
  - Check the return code when using this function! Look for MPI_SUCCESS

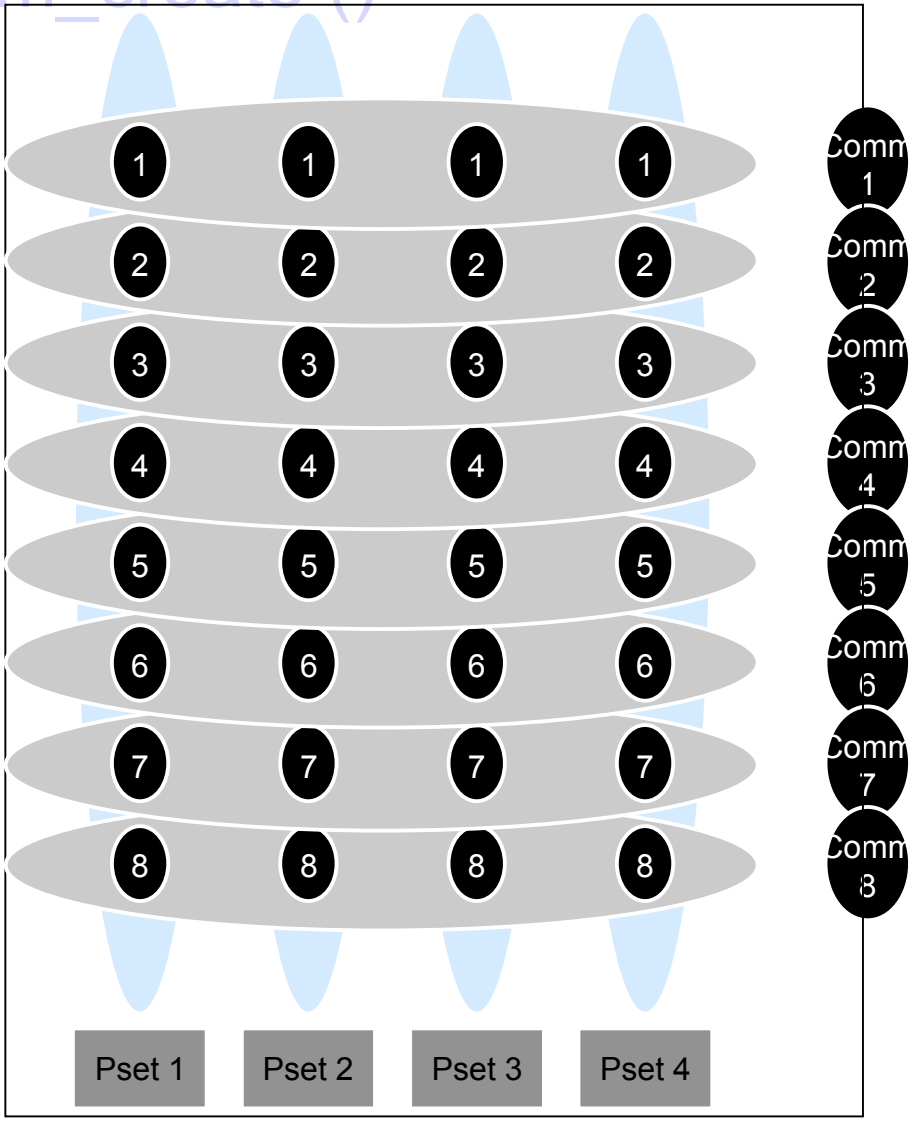
MPI on BG/P - MPIX_Pset_same_comm_create()

- **Creates communicators where the members share an I/O node**
- **Useful to maximize the number of I/O nodes used during I/O operations**
  - Node 0 in each of the communicators can be arbitrarily used as the “master node” for the communicator, collecting information from the other nodes for writing to disk.



MPI - MPIX_Pset_diff_comm_create ()

- **All nodes in the communicator have a different I/O node**
- **Rarely used without using same_comm() too**
  - Nodes without rank 0 in same_comm() sleep
  - Nodes with rank 0 in same_comm() would have a communicator created with diff_comm(). That communicator could be used instead of MPI_COMM_WORLD for communication/coordination of I/O requests



MPI on BG/P – Collectives

- **Currently optimized collectives:**
  - Broadcast (COMM_WORLD, rectangle, arbitrary)
  - (All)reduce (COMM_WORLD, rectangle, arbitrary)
  - Alltoall(v|w) (all comms, single threaded only)
  - Barrier (COMM_WORLD, arbitrary)
  - Allgather(v) (uses (async)bcast, reduce, or alltoall)
  - Gather (uses reduce)
  - Reduce_scatter (uses reduce, then scatterv)
  - Scatter (uses bcast)
  - Scatterv (uses alltoallv or bcast with an env var)

MPI on BG/P – Collectives

- **Use collectives whenever possible**
  - For example, replacing lots of sends/recvs with an alltoall(v)
  - Bad idea to implement collectives using your own point-to-point based algorithm
    - Too much overhead on point to point communications
    - Using MPI Send/Recv has message matching overhead
    - Can't take advantage of BG/P networks

MPI on BG/P – Collectives Options

- **Optimized collectives can be disabled if necessary**
  - DCMF_COLLECTIVE=0 or DCMF_COLLECTIVES=0
  - Disabling all can save application memory space, at the expense of performance (~10mb)
  - Specific collectives can be forced to use MPICH:
    - DCMF_{}=MPICH eg, DCMF_BCAST=MPICH
  - Specific collectives can attempt to use certain algorithms:
    - DCMF_ALLGATHER=ALLTOALLV **DCMF_SCATTERV=BCAST**
- **Generally unadvisable to force alternative algorithms.**
- **“Well-behaved” applications can also use:**
  - DCMF_SAFEALLGATHERV=Y
  - DCMF_SAFEALLGATHER=Y
  - DCMF_SAFESCATTERV=Y

MPI on BG/P – Our Collectives Strategies

- **Reduce/Allreduce:**
  - If the tree supports the operation and datatype and you are on COMM_WORLD, use the tree
  - else if you are on a rectangular communicator use a rectangular allreduce algorithm
  - else use a binomial algorithm
  - else use MPICH
- **Barrier:**
  - If you are on COMM_WORLD, use GI
  - else use binomial
  - else use MPICH

MPI on BG/P – Our Collectives Strategies

- **Bcast**
  - If COMM_WORLD use tree
  - Else if communicator is rectangular use an async rectangle protocol for small messages, then switch to synchronous
    - DCMF_ASYNCCUTOFF=8192
  - Else if communicator is irregular use an async binomial protocol for small messages, then switch to synchronous
    - DCMF_ASYNCCUTOFF=16384
  - Else use MPICH

MPI on BG/P – Our Collectives Strategies

- **Allgather(v)**
  - If treereduce and treebcast available and large message
    - Use bcast. If smaller message use reduce
  - If rectangular subcomm send a number of async bcasts, wait, repeat
    - DCMF_NUMREQUESTS=32 is default
  - If irregular subcomm send a number of async binom bcasts, wait, repeat
    - DCMF_NUMREQUESTS=32 is default
  - Else use alltoall
  - Else use MPICH

MPI on BG/P

- **Performance hints/suggestions**
  - Avoiding deadlock
  - Bad coding ideas
  - Touching buffers early
  - Mixing collectives and point-to-point
  - Flooding one node

MPI on BG/P – Avoid deadlocks

- **Talk before you listen.**
- **Illegal MPI code**
  - find it in most MPI books
- **BlueGene/P MPI is designed not to deadlock easily.**
  - It will likely survive this code.
- **This code will cause MPI to allocate memory to deal with unexpected messages. If MPI runs out of memory, it will stop with an error message**

CPU1 code:

MPI_Send (cpu2);

MPI_Recv(cpu2);

CPU2 code:

MPI_Send(cpu1);

MPI_Recv(cpu1);

MPI on BG/P – Send/Recv in Opposite Order

- **Post receives in one order, sends in the opposite order**
- **This is legal MPI code**
- **BlueGene/P MPI can choke if the sum of buffers is greater than the amount of physical memory**
  - Packet Pacing helps

    …but try to avoid doing this anyway

CPU1 code:

MPI_ISend(cpu2, tag_1);

MPI_ISend(cpu2, tag_2);

...

MPI_ISend(cpu2, tag_n);

CPU2 code:

MPI_Recv(cpu1, tag_n);

MPI_Recv(cpu1, tag_{n-1});

...

MPI_Recv(cpu1, tag_1);

MPI on BG/P – Touching buffers early

- **write send/receive buffers before completion**
  - results in data race on any machine
- **touch send buffers before message completion**
  - not legal by standard
  - BG/P MPI will survive it today
  - no guarantee about tomorrow
- **touch receive buffers before completion**
  - BG/P MPI will yield wrong results

```
req = MPI_Isend (buffer);
buffer[0] = something;
MPI_Wait(req);
```

```
req = MPI_Isend (buffer);
z = buffer[0];
MPI_Wait (req);
```

```
req = MPI_Irecv (buffer);
z = buffer[0];
MPI_Wait (req);
```

MPI on BG/P – MPI_Test memory leaks

- **Have to wait for all requests**
  - The standard requires waiting
  - Or looping until MPI_Test returns true
  - Otherwise, you are leaking requests
- **MPI_Test advances the message layer on each call**
- **We don't get much comm/compute overlap so just do the MPI_Wait instead of an MPI_Test.**

Code:

```
req = MPI_Isend( ... );

MPI_Test (req);

... do something else; forget about req ...
```

MPI on BG/P – Collectives mixed with point-to-point

- **On the ragged edge of legality**
- **BlueGene/P MPI works**
- **Multiple networks issue:**
  - Isend handled by torus network
  - Barrier handled by GI network
- **Try to avoid this**

CPU 1 code:

```
req = MPI_Isend (cpu2);
MPI_Barrier();
MPI_Wait(req);
```

CPU 2 code:

```
MPI_Recv (cpu1);
MPI_Barrier();
```

MPI on BG/P – Spamming one node

- **This is legal MPI code**
  - also ... bad idea
  - not scalable, even when it works
- **BlueGene/P MPI can run out of buffer space (packet pacing helps though)**
- **One (bad) solution – use SSend**
  - Forces synchronicity
  - Giant performance hit
- **Plenty of examples of this out there**
  - Don't write code such as this
  - Even if you think it should work

CPU 1 to n-1 code:

```
MPI_Send(cpu0);
```

CPU 0 code:

```
for (i=1; i<n; i++)
    MPI_Recv(cpu[i]);
```

MPI on BG/P – Spamming one node

- **Try multiple masters**
  - Need to find optimal master/submaster/worker arrangement
- **If funneling to one node for I/O**
  - Try MPI I/O
  - Use the communicator creation functions to optimize I/O usage

MPI IO

- **BG/P supports the full MPI IO implementation**
- **BG/P specific “device”, plus support for GFPS, PVFS**
- **Also use the MPIX_Pset_{}_comm_create routines**
- **Env vars for tuning**
  - BGLMPIO_COMM – Defines how data is exchanged on collective reads/writes. Default is 0 – Use MPI_Alltoallv. 1 uses MPI_Isend/Irecv
  - BGLMPIO_TUNEGATHER – Tune how offsets are communicated for aggregator I/O. Default is 1 – Use MPI_Allreduce. 0 uses two MPI_Allgather calls
  - BGLMPIO_TUNEBLOCKING – Tune how aggregate file domains are calculated. Default is 1 – Use the underlying file system’s block size and use MPI_ALLTOALLV to exchange information. 0 says evenly calculate file domains across aggregators and use MPI_Isend/Irecv to exchange the information

GPFS

- **Red paper/redbook coming soon for things to tune on your service node to improve GPFS performance**
- **Until then, use the MPIX_Pset_{} functions and do as much as you can at the app level for IO tuning**
- **We can help with specific IO questions too**

Questions?